Vulnerability Prediction Capability: A Comparison between Vulnerability Discovery Models and Neural Network Models

Yazdan Movahedi[a], Michel Cukier[a] , Ilir Gashi[b]

[a] Center for Risk and Reliability, University of Maryland, College Park, USA
{ymovahed, mcukier}@umd.edu
[b] Center for Software Reliability, City, University of London, London, U.K.
ilir.gashi.1@city.ac.uk

***Abstract*— In this paper, we introduce an approach for predicting the cumulative number of software vulnerabilities that is in most cases more accurate than vulnerability discovery models (VDMs). Our approach uses a neural network model (NNM) to model the nonlinearities associated with vulnerability disclosure. Nine common VDMs were used to compare their prediction capability with our approach. The different models were applied to vulnerabilities associated with eight well-known software (four operating systems and four web browsers). The models were assessed in terms of prediction accuracy and prediction bias. Out of eight software we analyzed, the NNM outperformed the VDMs in all the cases in terms of prediction accuracy, and provided smaller values of absolute average bias in seven cases. This study shows that NNMs are promising for accurate predictions of software vulnerabilities disclosures.**

Keywords*— *Vulnerability Discovery Model, Neural Network Model, Time Series, Vulnerability Discovery Process, Prediction, Software Reliability.

I. INTRODUCTION

Researchers have modeled new vulnerabilities disclosure trends using data from a variety of vulnerability databases. The goal of most of these studies is to find the model, which fits best the vulnerability disclosure process and, using that model, to predict the number of vulnerabilities that may be discovered for a given product [1]–[6]. Predicting the frequency of disclosures for vulnerabilities is useful for vendors of these products as well as the end-users as it helps them with resource allocation. In addition, such estimates can also provide useful information for evaluating the risk associated with a product that can be used by insurance companies [7].

Vulnerability discovery models (VDMs) were developed to predict future software vulnerabilities based on their historical behavior. Although VDMs are often accurate in terms of curve fitting, they might not perform well in prediction [8]. Indeed, VDMs are often not powerful enough to take the nonlinear nature of vulnerability disclosure into consideration.

In this paper, we introduce a nonlinear modeling approach based on neural networks to predict the total number of software vulnerabilities in 30-day time intervals. We compare the prediction capability of the neural network model (NNM) with nine commonly used VDMs. We applied the models to vulnerability data associated with four well known operating systems (OS) (Windows, Mac, Cisco IOS (the OS associated with Cisco), and Linux), as well as four well-known web browsers (Internet Explorer, Safari, Firefox, and Chrome).

Our work makes the following contributions:

- We introduce an approach using NNM to model the nonlinearities associated with vulnerability disclosure;
- We compare the capability of the NNM and nine VDMs in predicting the total number of software vulnerabilities in 30-day time intervals on eight well-known software (We predicted the vulnerabilities reported in years 2016, 2017, and 2018.);
- We show that the NNM outperforms the VDMs in all the cases in terms of prediction accuracy, and provides smaller values of absolute average bias in seven cases.

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the dataset used in the analysis. Section 4 lists the VDMs used in our analysis. Section 5 introduces the approach using an NNM to model discovery process of software vulnerabilities. Section 6 presents the results of using the dataset with the models for prediction. Section 7 discusses the main findings and some limitations. Finally, Section 8 presents conclusions and provisions for future work.

II. RELATED WORK

Over the years, many vulnerability discovery models (VDMs) have been developed to predict future software vulnerabilities based on their historical behavior.

The earliest effort at modeling software reliability was a Markov birth-death model [9]. A good overview of several software reliability growth models (SRGMs) that characterize the process of software defect-finding is provided in [2]. The earliest study on modeling the vulnerability discovery process [10] proposed the first VDM termed the Anderson Thermodynamic (AT) model. Rescorla [4], [5] proposed a VDM to estimate the number of undiscovered vulnerabilities. Alhazmi et al. [11] proposed the application of SRGMs to vulnerability

discover modeling. They also introduced a logistic VDM known as Alhazmi–Malaiya Logistic (AML) model, which assumes a symmetrical shape around the peak discovery rate value [6]. A Weibull distribution-based VDM was proposed by Kim [12]. Li et al. [13] empirically showed that, in comparison to other reliability models, a Weibull model is better for defect occurrence across a wide range of software systems.

Several studies applied existing VDMs or a modified versions of them to different types of software packages, such as OSs and web servers, to simulate the vulnerability discovery rate and predict the number of vulnerabilities that may potentially be present but not yet found [14]–[17]. Other studies tried to increase the accuracy of vulnerability discovery modeling by taking the skewness of the vulnerability data into consideration [8] or using the clustering techniques [18], [19], commonly used in social media studies [20] .

In recent years, some software vulnerability disclosure process models were developed using traditional time series models like Auto Regressive Moving Average (ARIMA) [21]. However, vulnerability disclosure data contain a lot of nonlinearity and thus traditional time series models might not be appropriate [22]. Pokhrel et al. [23] compared the modeling capability of linear and nonlinear time series for three OSs (i.e. Windows 7, Mac OS X, and Linux Kernel). They developed models based on ARIMA, Artificial Neural Network (ANN), and Support Vector Machine (SVM) settings.

III. Dataset

The dataset used in this paper was collected from the National Vulnerability Database (NVD) maintained by NIST, and collected using the same approach followed by [24]. We leveraged the vulnerability CVE IDs to compare the reporting date of each vulnerability in NVD with the dates in other public repositories on vulnerabilities[1]. We updated the reporting dates to the earliest date that a given vulnerability was publically known in any of the vulnerability databases they used.

We will analyze the reported vulnerabilities associated with four well-known OSs: Windows (1995-2018), Mac (1997-2018), Cisco IOS (the OS associated with Cisco) (1992-2018), and Linux (1994-2018), as well as four well-known web browsers: Internet Explorer (1997-2018), Safari (2003-2018), Firefox (2003-2018), and Chrome (2008-2018). These software have been selected because they are the most widely used and have the most vulnerabilities in the database. Figure 1 shows the detection frequency of all vulnerabilities associated with each software over time intervals of 30 days. We also plotted the 180-days moving average (MOVAVG) for each software to gain a better understanding of vulnerability detection trend. As is shown, the maximum value of MOVAVG for all cases occurred after 2015. For each software, the variable we used in this research is the cumulative number of vulnerabilities reported in 30-day time intervals. In other words, we divided the study period associated with a given software into intervals of 30 days, and counted the total number of vulnerabilities detected in each time interval.

For each software, we analyze all vulnerabilities reported for any of its versions. Thus, for each software, all

TABLE I. Number of Vulnerabilities per Software

OS	Windows	Mac	Cisco IOS	Linux
Total	3434	2908	698	5812
Train	2472	2081	522	3184
Test	962	827	176	2628
Web Browser	**IE**	**Safari**	**Firefox**	**Chrome**
Total	1862	994	1784	1906
Train	1289	701	1331	1229
Test	573	293	453	677

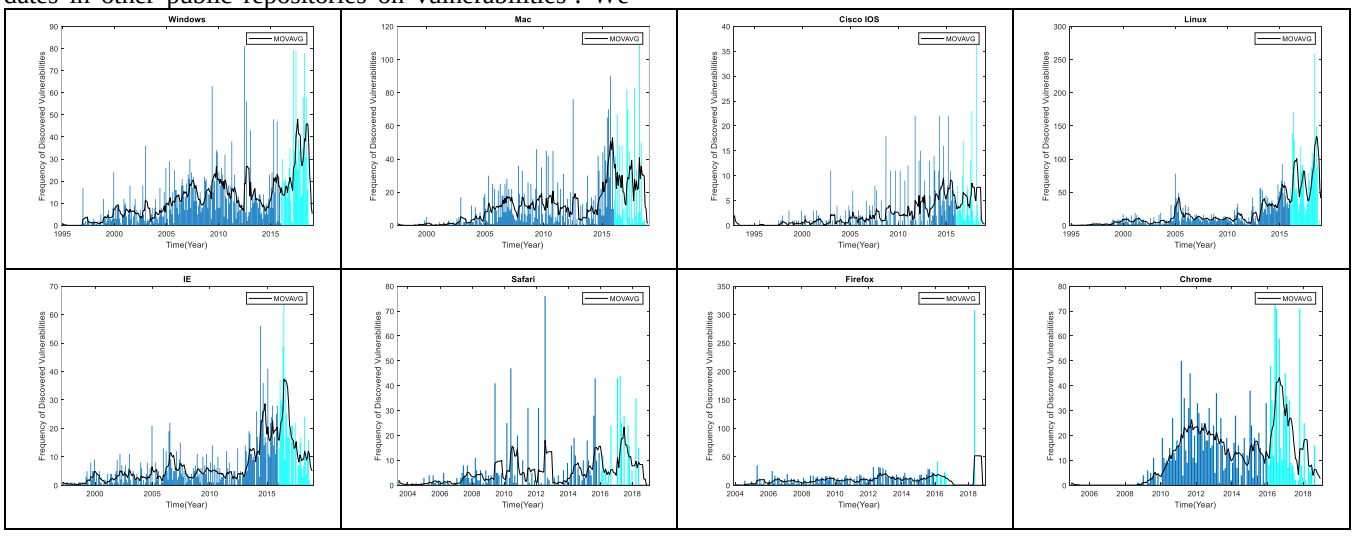

Fig. 1. Histogram of the number of detected vulnerabilities per 30 days together with its 180-days moving average for the studied OSs and Web browsers. The X-axis represents time (Year). The Y-axis shows the frequency of discovered vulnerabilities over 30 days time intervals. The dark and light colors show data associated with the training and test datasets (see Section VI).

[1] We looked at the following ones: http://www.cvedetails.com/, https://cxsecurity.com/, http://www.security-database.com/ and http://www.securityfocus.com/

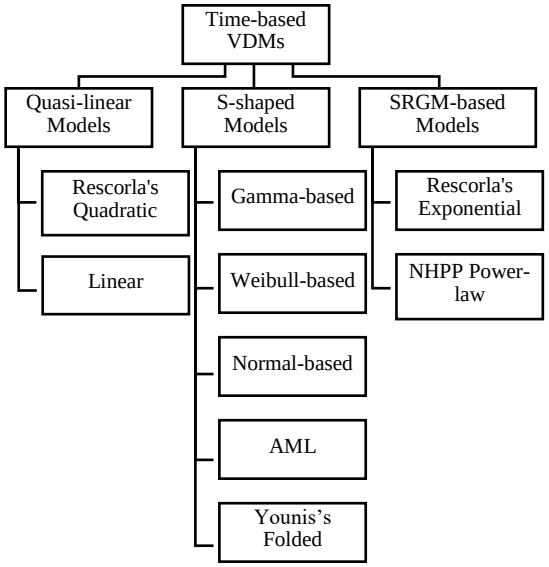

Fig. 2. Classification of Considered Time-based VDMs

the vulnerabilities reported for any of its versions were included. For instance, all the vulnerabilities reported for mac_os, mac_os_server, mac_os_x, and mac_os_x_server were put together to create a vulnerability database for Mac.

In addition, regarding our analysis, we divided the vulnerability dataset associated with each software into two groups; training and testing. The training data set consists of all the vulnerabilities reported before 2016. The testing data set consists of vulnerabilities reported in years 2016, 2017, and 2018. Table I represents the total number of vulnerabilities per software, as well as the number of vulnerabilities in the train and test datasets.

IV. Vulnerability Discovery Models (VDMs)

Vulnerability discovery models (VDMs) can be classified into two categories: time-based and effort-based VDMs. Time-based VDMs count the vulnerabilities of a given software as a function of calendar time while the effort-based VDMs, introduced by Alhazmi et al. [6], consider changes in environmental factors over lifetime of the software such as number of installations, share of installed base of a specific software, etc. (see [6] for more information regarding effort-based models). In this paper, we will only use time-based models since the data sources we have used only have data about vulnerability report dates, and not installations of the different software. Figure 2 shows a classification of time-based VDMs used in this research based on [16]. These models are the most common VDMs used in the literature. We considered at least one model from each class.

S-shaped VDMs divide the process of vulnerability discovery into three phases as shown in Figure 3. Phase 1 represents the learning phase, which starts from the introduction of the software and continues until the beginning of the period referred to as "Sustained Growth" as a consequence of increasing popularity of the software [8]. During the learning phase, the vulnerability discovery intensity function is an increasing function. Phase 2 or the linear phase is the period when most of the vulnerabilities are expected to be detected.

TABLE II. Table of Models and their Equations

Model	Equation
NHPP Power-law [18]	$\Omega(t) = (\beta^{-\alpha}).t^{\alpha}$
Gamma-based VDM [8]	$\Omega(t_0) = \int_{t=0}^{t_0} \frac{\gamma}{\Gamma(\alpha)\beta^{\alpha}} t^{\alpha-1} e^{-\frac{t}{\beta}} dt$
Weibull-based VDM [12]	$\Omega(t) = \gamma\{1 - e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\}$
AML VDM [6]	$\Omega(t) = \frac{B}{BCe^{-ABt}+1}$
Normal-based VDM [8]	$\Omega(t) = \frac{\gamma}{1+e^{-\frac{(t-\mu)}{s}}}$
Rescorla Exponential (RE) [5]	$\Omega(t) = \gamma(1 - e^{-\lambda t})$
Rescorla Quadratic (RQ)[5]	$\Omega(t) = \frac{At^2}{2} + Bt$
Younis Folded (YF) [25]	$\Omega(t) = \frac{\gamma}{2}\{\text{erf}\left(\frac{t-\tau}{\sqrt{2\sigma}}\right) + \text{erf}\left(\frac{t+\tau}{\sqrt{2\sigma}}\right)\}$
Linear Model (LM) [26]	$\Omega(t) = At + B$

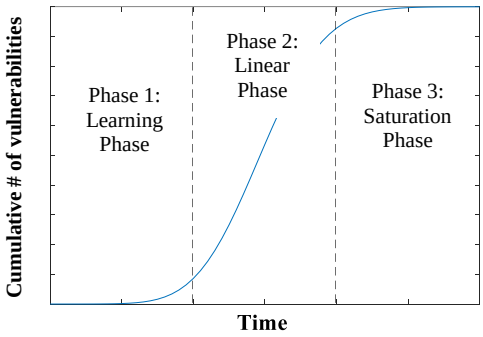

Fig. 3. Three Phases for S-shaped Models

The intensity function of this phase is constant. Phase 3 or the saturation phase is the period when most of the vulnerabilities have been discovered [16]. The vulnerability discovery intensity function for the saturation phase is decreasing. This phase will not appear as long as a significant number of vulnerabilities are still undetected.

The five S-shaped VDMs used in the paper based upon their capability in modeling skewed data can be classified as follows: two right-skewed distributions (Gamma-based VDM, Younis Folded VDM), one flexible-skewed distribution (Weibull-based VDM), and two symmetrical distributions (Alhazmi–Malaiya Logistic (AML) model and Normal distribution-based model). These VDMs are selected because they are the most well-known VDMs used in modeling the vulnerability discovery process [8].

In addition, we have also included four non-S-Shaped VDMs: Rescorla Exponential (RE) model, Rescorla Quadratic (RQ) model, NHPP Power-law model, and Linear model (LM). More information about the Rescorla models and the linear model can be found in [5] and [26], respectively. When modeling the mean cumulative number of failures $\Omega(t)$ for software reliability evaluations, models derived from a nonhomogeneous Poisson process (NHPP) are often used. Allodi [27] showed that the discovered vulnerabilities may follow a Power-law distribution. The model used in this paper was applied on vulnerability data as a VDM in [24] [18]. The main assumption of this model

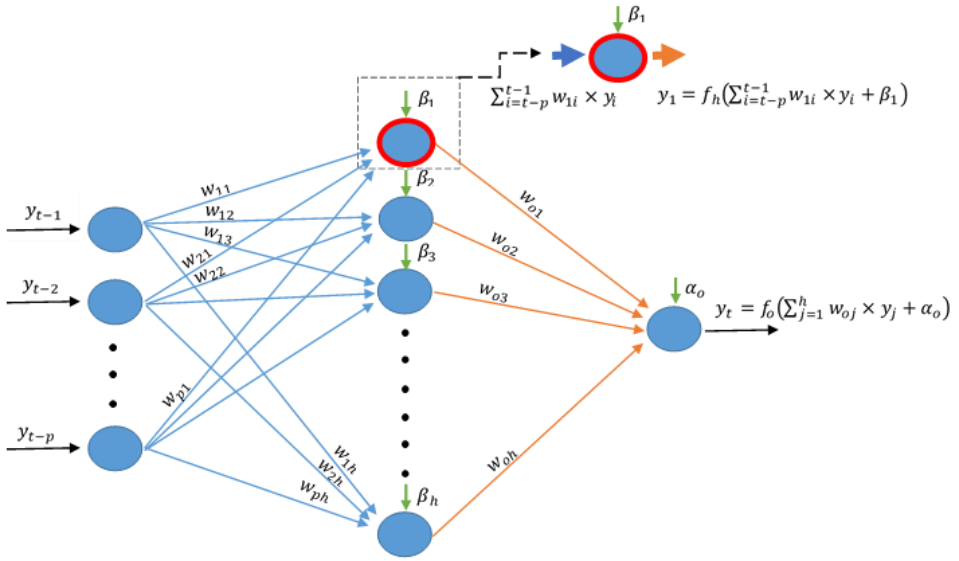

Fig. 4. The NNM Architecture Used for Our Study

is that the number of discovered vulnerabilities follows a nonhomogeneous Poisson process. In addition, in NHPP-based software reliability growth models (SRGMs), the intensity function ($\omega(t) = dE[\Omega(t)]/dt$) is considered to be a monotonic function [28].

The equations of all the discussed models are presented in Table II.

V. Neural Network Model (NNM)

Neural network models (NNMs) consist of a set of algorithms for modeling and recognizing patterns. NNMs have been widely used for predicting data with sequential time series data such as monthly electricity demand of a city or stock price [22], [29], [30]. Unlike VDMs, NNMs are capable of integrating the nonlinearity that exist in noisy time series data. In addition, NNMs are not built upon assumptions regarding the form of the basic model since they are completely data driven models. In other words, NNMs are flexible nonlinear data driven models with powerful prediction power. Data driven models are very useful for the cases, where there is not any appropriate theoretical guidance to explain data generation process. It has been empirically shown that NNMs are capable of predicting both linear and nonlinear time series of different forms [31].

In this study, to predict the number of discovered vulnerabilities getting over time for a given software, we use a feedforward NNM, which is the most widely used neural network [22]. Feedforward NNMs accept a fixed number of inputs at a time and generate one output. We assume that the number of future vulnerabilities depend on the number of vulnerabilities disclosed over the past periods (lags).

In this study, we use a single hidden-layer NNM for one step-ahead forecasting. According to [32], a single hidden layer NNM is capable of approximating any non-linear function with arbitrary precision. Figure 4 shows the structure of the NNM used in our study. Our feedforward NNM consists of three layers called input, hidden, and output. Each layer is a collection of neurons (nodes) where the connections are governed by the corresponding weights. Data have been fed through the input layer, and then they pass through the one or more hidden layers, and the final outcome is provided by the output layer.

To predict the present value, several past observations are used. In other words, the inputs are a p-element subset of the set $\{y_{t-p}, \ldots, y_{t-2}, y_{t-1}\}$; and y_t is the output or the total number of vulnerabilities reported in period t. Equations 1 and 2 show the formulas associated with the input and output values of the hidden layer, respectively. For the output layer, the input and output values are represented by Equations 3 and 4, respectively.

$$I_j = \sum_{i=t-p}^{t-1} w_{ji} \times y_i + \beta_j \quad (j = 1, \ldots, h), \tag{1}$$

$$y_j = f_h(I_j) \quad (j = 1, \ldots, h), \tag{2}$$

$$I_o = \sum_{j=1}^{h} w_{oj} \times y_j + \alpha_o \quad (o = 1), \tag{3}$$

$$y_t = f_o(I_o) \quad (o = 1), \tag{4}$$

I denotes the input; y denotes the output; p and h are the number of input and hidden layer nodes, respectively; w_{ji} represents the connection weights of the input and hidden layers; and w_{oj} denotes the connection weights of the hidden and output layers. The bias values of the hidden and output layers are respectively shown by β_j and α_o, and are always between -1 and 1. f_h and f_o are the non-linear activation functions associated with the hidden and the output layers, respectively. As the hidden layer activation function, we used a hyperbolic tangent function since it is the function that most widely used [22].

The initial step in designing a NNM is to determine the optimal number of input nodes (lags) and hidden layer nodes. Based on the literature, there is no systematic approach [22]; the most common way of identifying the

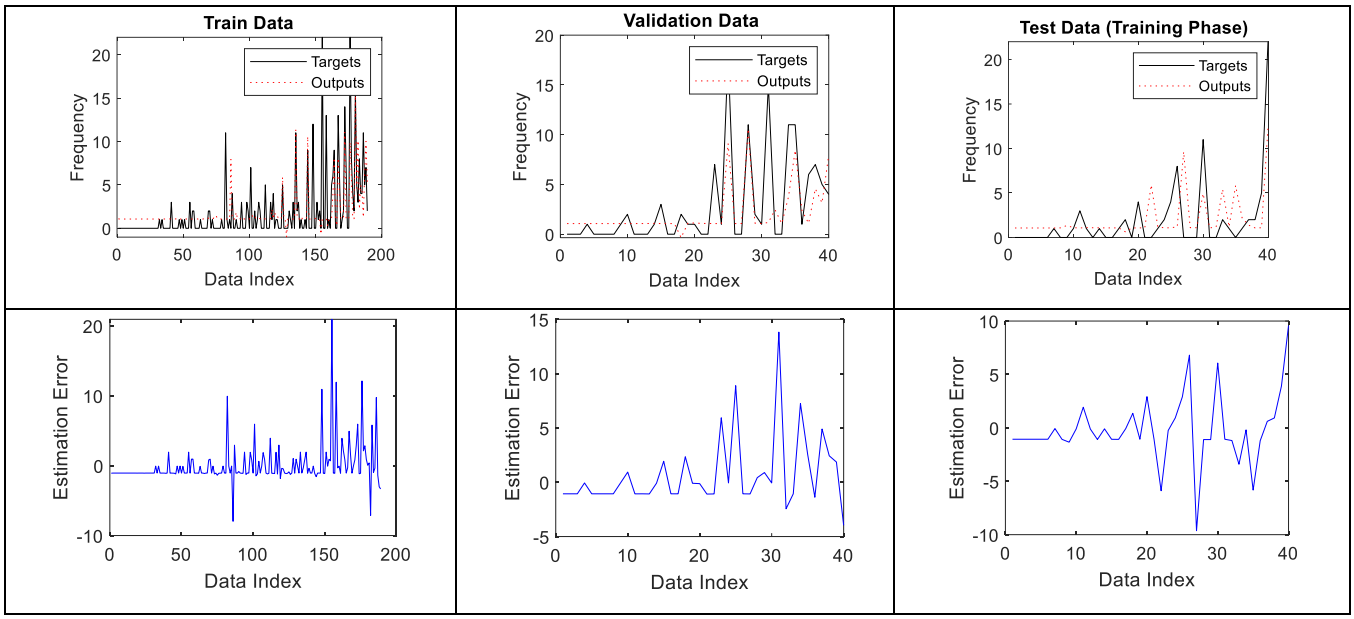

Fig. 5. Plots associated with training phase of the proposed NNM for Cisco IOS

appropriate number of the nodes (input and hidden) is via trial and error based upon finding the minimum mean square error (MSE) of a subset of the training data, used for initial test and validation [33]. We followed the optimization approach based upon the algorithm (ADE-BPNN) introduced in [22] to identify the optimal number of inputs (lags) and the number of hidden nodes the time series associated with each software. In [22], it is shown that using ADE-BPNN improves prediction accuracy relative to basic NNMs, autoregressive integrated moving average model (ARIMA), and other hybrid models for time-series data. We evaluated up to 50 hidden nodes for each time series and chose the number of hidden nodes that minimized the MSE. We started with statistically significant lags derived from the process of evaluating the partial autocorrelation function (PACF) associated with each time series. In time series analysis, PACF gives the linear partial correlation of a time series with its own lagged values and evaluated [34]. However, we cannot only rely on the lags we found from the PACF since, in such case, the selection of inputs would be merely based on the identification of a linear model, while the goal for using NNM is to capture non-linear correlations, as well. A very good review of existing input selection methods for NNMs is provided in [35].

The NNM developed in this paper was programmed using Matlab R2018a. For each software, we began our analysis by dividing the vulnerability dataset into two groups; training and testing. We used identical data points for both modeling approaches (VDMs and NNM). The training dataset consists of all the vulnerabilities reported before 2016. The testing data set consists of vulnerabilities reported in years 2016, 2017, and 2018. NNM training is a complex nonlinear optimization problem. Thus, there is the possibility to get trapped in local minima of the error surface. To avoid getting poor results, the training process should be repeated several times with different random starting weights and biases [31].

We set the maximum training number equal to 500 epochs. Epoch stands for the total number of times a given dataset is utilized for training and shows the number of times the weights in a network were updated [36]. Since model optimization in deep learning algorithms is done using the gradient decent method [37], it makes sense to pass the learning dataset through the network multiple times accordingly to update the weights and achieve a more accurate forecasting model [36]. We used the Levenberg-Marquardt (LM) method as our learning function. The activation function of the hidden and output layers are the *tansig* and *purelin* functions, respectively. To avoid overfitting/over training, for each software, we employed a cross validation method by splitting our dataset into three subgroups of training data (70%), validation data (15%), and test data (15%); and checked the validation performance of the trained network via metrics provided by Matlab Neural Network toolbox such as gradient decent (*gradient threshold*=1.00e-4) and maximum number of validation checks (*max_fail*=100). These metrics served as stop conditions of the training phase. Whenever the parameters of the network under training met any of these thresholds, the training process was stopped. Figure 5 shows the plots associated with the training phase of the NNM for Cisco IOS. As it is shown, based on the error plot of train data, there is no sign of overfitting. Moreover, the plots associated with the test subgroup of training data shows that the model performed well in the training phase.

VI. Results

We used the nine VDMs and one NNM for the discovery process of vulnerabilities in eight well-known software (four OSs and four web browsers). The VDMs were fitted to the datasets using a non-linear regression method described in [16].

The analysis of the prediction capability started by dividing the data into two groups of training and test data. Both the VDMs and the NNM use a dataset that includes all vulnerabilities reported for all versions of a given software. The training period starts from the time when the first vulnerability associated with a given software was discovered and continues until 12/31/2015. We calculated the predictions for the years 2016, 2017, and 2018. As it is shown in Figure 1, the dark and light bars show the data associated with training and test datasets, respectively. We

split the vulnerability data into intervals of 30 days as is common in the vulnerability analysis literature [8], [15], [16].

For the VDMs, during the training period, the training data was used to estimate model parameters. To avoid overfitting, 10-fold cross validation was also conducted on the training data. The estimated final values for each interval produced by the nine models were compared with the actual number of vulnerabilities to calculate the prediction accuracy. For the NNM, for each software, we used the training data to train the NNM. Using the trained NNM, we predicted the next values for the intervals in the prediction period. The prediction accuracy is based on the comparison between the obtained estimation and the actual number of vulnerabilities.

For the training part, for VDMs, we applied the Chi-square (χ^2) goodness of fit test [16] to assess how well each model fits the training datasets. The χ^2 statistic is calculated using the following equation:

$$\chi^2 = \sum_{i=1}^{N1} \frac{(S_i - E_i)^2}{E_i} \tag{5}$$

where S_i and E_i are the simulated and expected observed values at i^{th} time point, respectively. N1 is the number of observations in the train dataset (the time blocks used for simulation). For the fit to be acceptable, the corresponding χ^2 critical value should be greater than the χ^2 statistic for the given alpha level and degrees of freedom. We selected an alpha level of 0.05. The null hypothesis indicates that the actual distribution is well described by the fitted model. Hence, if the p-value of the χ^2 test is below 0.05, then the fit will be considered unsatisfactory. A p-value closer to 1 indicates a better fit. For each VDM, before evaluating its predictive capability, first we check whether it is statistically sound or not. If it is not sound (p-value<0.05), we neglect that model in our evaluation process.

For the training part, for the NNM, out of the models trained with different number of lags, the optimal analytical model was selected based on the MSE value. Finally, for each software, the best selected analytical model was used to make the prediction for the testing data set (the vulnerabilities reported in 2016, 2017, and 2018). In this study, regarding the NNMs, we just reported the results associated with the best NNM since it was not possible to include all the trained NNMs per software with different combination of lags due to space limit.

We calculated two normalized predictability measures, average error (AE) and average bias (AB) [8]. AE is a measure of how well a model predicts throughout the test phase, and AB indicates the general bias of the model which assesses its tendency to overestimate or underestimate the number of discovered vulnerabilities. AE and AB are defined as:

$$AE = \frac{1}{N2} \sum_{t=1}^{N2} \left| \frac{\Omega_t - \Omega}{\Omega} \right| \tag{6}$$

$$AB = \frac{1}{N2} \sum_{t=1}^{N2} \frac{\Omega_t - \Omega}{\Omega} \tag{7}$$

where N2 is a total number of time points (one per 30 days) over the prediction period, and Ω is the actual number of total vulnerabilities at time t, whereas Ω_t is the estimated number of total vulnerabilities at time t.

In addition, for the VDMs, we report ΔAE_i, which represents the percentage of difference between the AE of the i-the model and the model with minimum AE.

$$\Delta AE_i = (AE_i - AE_{min}) * 100 \tag{8}$$

where AE_i is the AE of the i-th model, and AE_{min} is the lowest AE one obtains among the set of models examined (i.e., the best model).

The root mean square error (RMSE) is another metric often used to calculate fitting errors. However, Mentaschi et al. [38] showed that for some applications (e.g., high fluctuation of real data) the lower values of RMSE are not always a reliable indicator of the accuracy of simulations. Hence, a corrected estimator HH was proposed by Hanna and Heinold [39]:

$$HH = \sqrt{\frac{\sum_{i=1}^{N2}(S_i - O_i)^2}{\sum_{i=1}^{N2} S_i O_i}} \tag{9}$$

where S_i is the i^{th} simulated data, O_i is the i^{th} observation (test data) and N2 is the number of observations in test dataset (the time blocks used for simulation). The closer to zero HH is, the more accurate the model.

Tables III- IV present the values of AE, AB, HH, ΔAE_i, and p-value (we used * to show the models with p<0.05) for the cases we analyzed per model (VDMs and NNM), respectively. We also used the term "NS" as the ΔAE_i value of the models with p<0.05, which stands for Not Satisfactory. AB can be positive (for overestimation) or negative (for underestimation), while AE is always positive. In each case, we first found the best VDMs by comparing their prediction accuracy and then compared the accuracy of those models with the NNM. In other words, for the VDMs, the model that has the smallest value of AE was selected as having the best prediction capability and is highlighted in yellow. In addition, the VDMs with $\Delta AE_i < 2$ were also selected as the best forecasting VDMs, which, show similar prediction capability compared to the best model (the model/models with $\Delta AE_i = 0$). In addition, the normalized error values ($(\Omega_t - \Omega)/\Omega$) associated with each case are plotted in Fig. 6. As it is shown, the models with less fluctuations yield higher accuracy.

Based on the results provided by tables III-IV, in terms of prediction accuracy (AE and HH), the NNM led to most accurate results in all of the eight software we analyzed. To be more precise, for Windows, the NNM's average error (AE) is 2.6%, and, at least, is 1.1% smaller than the AEs associated with the best VDMs, which were Power-law and RQ. For Mac, the NNM outperforms the best VDMs

TABLE III. PREDICTION ACURRACY FOR OSS (VDMS & NNM)

	Windows					*Mac*				
	AE	***AB***	***HH***	***%ΔAE$_i$***	***p-value***	***AE***	***AB***	***HH***	***%ΔAE$_i$***	***p-value***
Gamma	0.063	-0.061	0.095	2.568	0.686	0.218	-0.218	0.263	14.373	0.193
Weibull	0.091	-0.091	0.131	5.454	0.936	0.233	-0.233	0.287	15.880	0.703
AML	0.138	-0.138	0.187	NS	0.011*	0.278	-0.278	0.351	20.401	0.301
Normal	0.138	-0.138	0.187	NS	0.011*	0.278	-0.278	0.351	20.401	0.301
Power-law	0.037	0.025	0.040	0.000	0.466	0.074	-0.074	0.080	0.000	0.138
RE	0.106	0.106	0.100	NS	0.026*	0.024	0.017	0.037	NS	0.000*
RQ	0.039	0.030	0.042	0.205	0.888	0.082	-0.082	0.090	0.776	0.193
YF	0.114	-0.114	0.158	7.687	0.200	0.256	-0.256	0.320	18.195	0.193
LM	0.192	-0.192	0.235	NS	0.001*	0.313	-0.313	0.392	NS	0.000*
NNM	0.026	-0.018	0.036	NA	NA	0.021	-0.019	0.025	NA	NA
	Cisco IOS					*Linux*				
	AE	***AB***	***HH***	***%ΔAE$_i$***	***p-value***	***AE***	***AB***	***HH***	***%ΔAE$_i$***	***p-value***
Gamma	0.018	0.001	0.025	0.000	0.900	0.268	-0.268	0.353	7.804	0.640
Weibull	0.019	0.006	0.028	0.106	0.901	0.267	-0.267	0.352	7.751	0.640
AML	0.076	0.076	0.088	5.776	0.660	0.272	-0.272	0.366	8.249	0.335
Normal	0.076	0.076	0.088	5.775	0.660	0.272	-0.272	0.366	8.248	0.335
Power-law	0.019	0.006	0.028	0.114	0.232	0.267	-0.267	0.352	7.734	0.640
RE	0.131	0.131	0.148	11.304	0.142	0.190	-0.190	0.239	0.000	0.506
RQ	0.154	-0.154	0.172	NS	0.000*	0.278	-0.278	0.369	8.851	0.222
YF	0.092	0.092	0.108	7.427	0.941	0.240	-0.240	0.313	4.991	0.109
LM	0.422	-0.422	0.567	NS	0.000*	0.441	-0.441	0.643	NS	0.000*
NNM	0.017	-0.014	0.024	NA	NA	0.041	0.026	0.044	NA	NA

TABLE IV. PREDICTION ACCURACY FOR WEB BROWSERS (VDMS & NNM)

	IE					*Safari*				
	AE	***AB***	***HH***	***%ΔAE$_i$***	***p-value***	***AE***	***AB***	***HH***	***%ΔAE$_i$***	***p-value***
Gamma	0.233	-0.233	0.272	9.251	0.624	0.156	-0.156	0.201	12.634	0.245
Weibull	0.233	-0.233	0.272	9.205	0.624	0.187	-0.187	0.245	15.704	0.739
AML	0.157	-0.157	0.175	NS	0.003*	0.231	-0.231	0.304	NS	0.048*
Normal	0.157	-0.157	0.175	NS	0.003*	0.231	-0.231	0.304	NS	0.048*
Power-law	0.233	-0.233	0.271	9.198	0.624	0.030	0.026	0.037	0.000	0.378
RE	0.149	-0.149	0.164	0.818	0.403	0.133	0.133	0.141	NS	0.012*
RQ	0.232	-0.232	0.270	9.096	0.624	0.041	0.040	0.047	1.087	0.549
YF	0.141	-0.141	0.155	0.000	0.285	0.211	-0.211	0.278	18.093	0.986
LM	0.384	-0.384	0.501	NS	0.012*	0.204	-0.204	0.248	NS	0.009*
NNM	0.066	-0.066	0.072	NA	NA	0.027	0.024	0.034	NA	NA
	Firefox					*Chrome*				
	AE	***AB***	***HH***	***%ΔAE$_i$***	***p-value***	***AE***	***AB***	***HH***	***%ΔAE$_i$***	***p-value***
Gamma	0.051	0.035	0.066	0.151	0.378	0.281	-0.281	0.367	0.000	0.735
Weibull	0.049	0.031	0.064	0.030	0.307	0.317	-0.317	0.422	3.578	0.735
AML	0.081	-0.081	0.112	NS	0.012*	0.307	-0.307	0.405	2.569	0.641
Normal	0.081	-0.081	0.112	NS	0.012*	0.307	-0.307	0.405	2.569	0.641
Power-law	0.069	0.067	0.089	2.015	0.307	0.167	0.167	0.191	NS	0.000*
RE	0.161	0.161	0.174	11.231	0.115	0.364	0.364	0.383	NS	0.000*
RQ	0.096	0.096	0.113	4.696	0.193	0.077	0.077	0.102	NS	0.000*
YF	0.049	-0.032	0.069	0.000	0.150	0.304	-0.304	0.402	2.269	0.117
LM	0.101	-0.101	0.123	5.204	0.087	0.302	-0.302	0.374	NS	0.000*
NNM	0.037	-0.012	0.051	NA	NA	0.033	-0.013	0.038	NA	NA

(Power-law, and RQ) by having 5.3%, and 6.1% smaller average errors, respectively. For Cisco IOS, this difference is at least 0.1%. Linux and Chrome are two of the cases where the NNM provides far better predictions than those from VDMs by being 14.9%, and 24.8% more accurate. For IE, this difference is at least 7.5%. The average error of the NNM for Safari is 0.3% and 1.4% smaller than those from the best VDMs (Power-law and RQ).

For Firefox, the NNM improved the predictions by 1.2%, 1.4%, and 1.2% compared to YF, Gamma, and Weibull VDMs, respectively. For Chrome, the VDM with smallest AE is not statistically sound from the training part. So, we opt for the next VDM with p-value>0.05 and smallest AE, which is Gamma. In this case, the NNM accuracy improvement is 24.8%. Overall, the highest differences in prediction accuracy between the NNM and the VDMs were found in Chrome (24.8%), Linux (14.9%), IE (7.5%), and Mac (5.3%), respectively.

Another factor, which plays an important role in model selection is the model tendency to overestimate or underestimate the results. In this research, we provided the average bias values (AB) as well as the visual fluctuation trend of normalized prediction errors (Figure 6). Based on tables III-IV, out of eight software we analyzed, in terms of magnitude of error, the NNM outperformed the VDMs in seven cases by having smaller |AB| values. Only for Cisco IOS, the absolute value of bias provided by the selected VDMs was at most 1.3% smaller than the one resulted from the NNM. For Windows, Mac, Linux, IE, Safari, Firefox, and Chrome, the bias magnitudes provided by the NNM were smaller than those from the best VDMs (in each case, we considered the best VDM, which had smallest |AB|) by

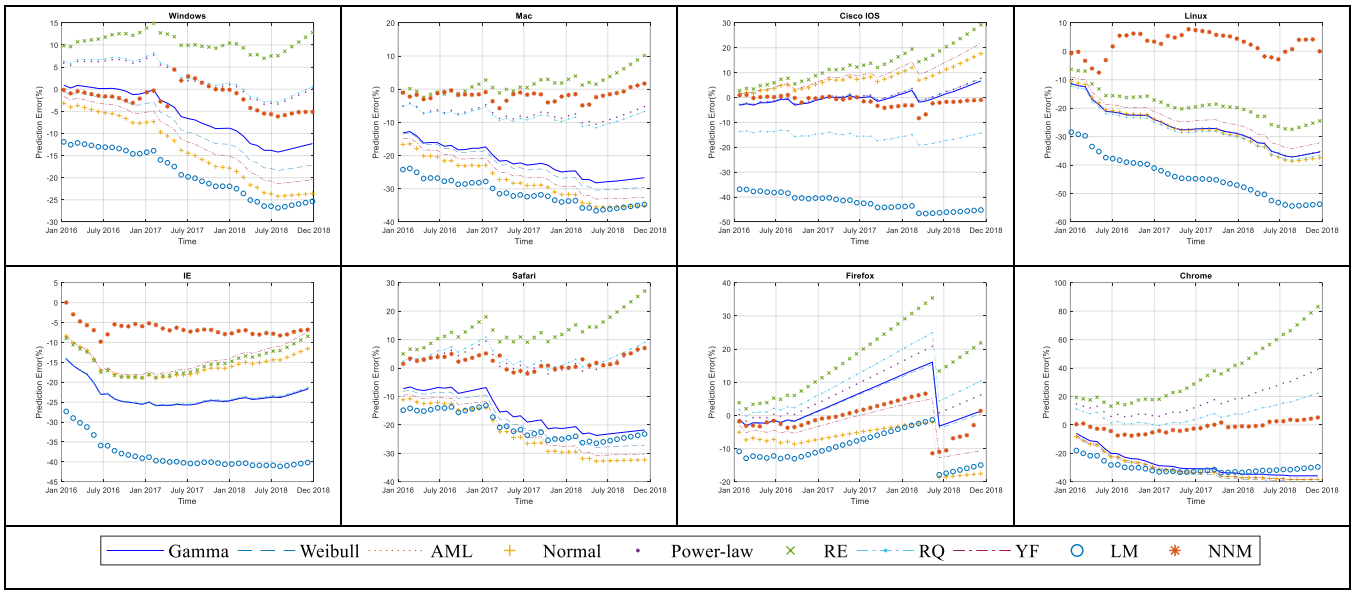

Fig. 6. Prediction errors for OSs and web browsers. The X-axis indicates time (Year). The Y-axis represents normalized prediction error values $((\Omega_t - \Omega)/\Omega)$.

0.7%, 5.5%, 16.4%, 7.5%, 0.2%, 1.9%, and 26.8%, respectively. Overall, in terms of accuracy, out of the eight cases we analyzed, the NNM outperformed VDMs in all the cases. Besides, in terms of magnitude of bias, the NNM led to smallest bias values in seven cases.

VII. Discussion and Limitations

In terms of prediction accuracy (AE and HH), considering the OSs and web browsers, the NNM led to more accurate results than the best selected VDMs in all the cases. Considering only VDMs, the Power-law VDM was selected as the best model in four cases out of eight. The Gamma and RQ VDMs were each best compared with other models in three cases out of eight. The Weibull, RE, and YF VDMs were each best compared with other models in two cases out of eight cases we analyzed.

In terms of overall magnitude of bias (i.e., absolute value of AB), out of the eight cases we analyzed, the NNM provided smaller absolute values of bias in seven cases compared to the best VDMs. Only for Cisco IOS, the absolute value of bias provided by the Gamma VDM (0.001) was smaller than the one resulted from NN (-0.014).

We believe that the final decision, in equal accuracy conditions, in terms of bias, is up to the researcher to choose the best model based upon his/her priorities. However, from a security point of view, it is better to choose a model, which provides more conservative prediction results. In the current study, among the models that were selected as the best predictors, only two NNMs (Linux and Safari) provided overestimated results. Other selected NNMs underestimated the number of vulnerabilities. It can also be easily inferred from Figure 6, where for Linux and Safari most of the prediction points associated with the NNMs are located over the X=0 axis.

In two cases (i.e. Linux and Chrome), the difference between the prediction accuracy of the NNM and the best relative VDM is very high (i.e. 14.9% and 24.8%, respectively). We believe that the reason behind these high value of differences is not associated with noise, since we took care of the noise problem in the data by using a cross validation technique. It is possible to assume that, for these three cases, the NNM was able to detect another playing factor in the relative vulnerability discovery processes that the analytical VDMs were not [40].

Overall, we believe that the NNM's better performance compared to VDMs comes from the capability of the NNM in predicting the nonlinearity nature of the vulnerability disclosure time series. In addition, most VDMs consider the vulnerability discovery process as a pure S-shaped curve or a function with a monotonic intensity function with constant total number of vulnerabilities. However, the number of vulnerabilities associated with a given software may change as newer versions are released. Additionally, VDMs and traditional time-series functions only use one set of parameters for estimation. On the other hand, NNMs due to having multilayer perceptron structure, having multiple neurons per layer, and using different set of parameters per neuron provide a more complicated structure for prediction. Of course, the specific validation method we used to avoid being trapped by overfitting in the learning phase is another advantage of using NNMs.

There are several limitations to our work that prevent us from making more general conclusions. The main limitation is with regard to using reported published date of vulnerabilities as their detection date. Vulnerabilities usually get detected earlier by malicious users than the time they are officially published. To make sure that this estimate is as close as possible to the actual date the vulnerability is known to the world, we looked at different vulnerability sources and opted for the earliest date reported for a vulnerability. Better estimates can be obtained if we have more accurate proxies for calculating attacker effort and more precise times on when a vulnerability is discovered and reported (for example, in the dark web), rather than when it is reported in a public vulnerability database. However, obtaining this data is difficult: data in the dark web is highly unstructured and very difficult to add meaning to what is mined.

Another limitation is with regard to the way we merged all vulnerabilities reported for all versions of a given software to have enough data for training the models. There are papers that apply VDMs to the vulnerabilities related to separate versions of software (e.g. Mac OS X) [16], [41] as well as studies that consider all versions of a software together [8], [18]. The first group assumes that each piece of software is an independent and well-defined product, but detecting the sources of dependency in vulnerability data is not an easy task.

NNMs, unlike analytical models, are not mathematically tractable and cannot be easily interpreted by humans. However, they are capable of detecting the mechanisms that might be missed by the analytical models. Therefore, they can be used a guide for modifying those models. In this research, we showed that more accurate predictions are also possible using NNMs [40].

VDMs assume that the time between failures represents total usage time of that product. What we are using is calendar time, which may not be a good proxy for usage. Crucially the difference in security is the difficulty in estimating the "attacker effort" - the total amount of time that an attacker spends in finding a vulnerability - which is something that is not needed for reliability (we assume the users accidentally encounter faults that lead to failures, hence usage time is a good enough proxy for time between failures). A useful discussion of this is given in [42]. Attacker effort is something that is very difficult to estimate and quantify. The purpose of our research is hence to make as good a use as possible of the publically available security data to help with decision making. But at the same time to be clear about the limitations on what we can conclude from this analysis. The best we can say from the analysis we present is "the total number of vulnerabilities that will be reported in the NVD over an interval t for product x is y with confidence z". And we show that we can do this prediction better with NNM than with VDMs for four of the largest and most commonly used operating system and web browser families. For some decision makers this may be a valuable piece of additional information, which they can use in conjunction with data they have from their own installations, when deciding on operating systems and/or web browser, and provisioning of security support services to deal with new vulnerabilities.

VIII. Conclusion and Future Work

In this paper, we compared the capabilities of nine common vulnerability discovery models (VDMs) with a nonlinear neural network model (NNM) in terms of predicting the number of future vulnerabilities over a prediction period of three years. We applied the mentioned models to vulnerability data associated with four well known OSs and four well-known web browsers. The models were assessed in terms of prediction accuracy and prediction bias. The results showed that the NNM outperformed the VDMs in all the cases in terms of prediction accuracy. In terms of overall magnitude of bias, out of the eight cases we analyzed, the NNM provided the smallest absolute values of bias in seven cases compared to the best VDMs. This study shows that neural networks are promising for accurate predictions of the total number of publically reported software vulnerabilities over time.

For future work, we plan to find the reason behind the observed gap between prediction capabilities of the NNMs versus VDMs, specifically for the two software that we found considerable difference. We will try to investigate whether current VDMs are missing a mechanism associated with the process of vulnerability discovery within their mathematical structure. We also plan to investigate other nonlinear model structures using machine learning algorithms. Among them are Recurrent Neural Network (RNN) models, used for prediction time series, which may perform better than NNMs at modeling dependencies between two points in a sequence. Generally, in NNMs, we have to choose the length of the input (number of inputs) beforehand. Then, it is not possible to learn functions that depends on the inputs that happened a long time ago. This problem could be solved by having an RNN, which can theoretically store information from arbitrarily long time ago.

IX. Acknoledgements

This research is supported by NSF Award #1223634, and the UK EPSRC project D3S (Diversity and defence in depth for security: a probabilistic approach) and the European Commission through the H2020 programme under Grant Agreement 700692 (DiSIEM).

References

[1] H. Okamura, M. Tokuzane, and T. Dohi, "Optimal Security Patch Release Timing under Non-homogeneous Vulnerability-Discovery Processes," presented at the 20th International Symposium on Software Reliability Engineering, 2009, pp. 120–128.

[2] M. R. Lyu, Ed., *Handbook of software reliability engineering*. Los Alamitos, Calif. : New York: IEEE Computer Society Press ; McGraw Hill, 1996.

[3] J. A. Ozment, "Vulnerability discovery & software security," University of Cambridge, 2007.

[4] E. Rescorla, "Security holes... Who cares?," presented at the USENIX Security, 2003.

[5] E. Rescorla, "Is finding security holes a good idea?," *IEEE Secur. Priv. Mag.*, vol. 3, no. 1, pp. 14–19, Jan. 2005.

[6] O. H. Alhazmi and Y. K. Malaiya, "Quantitative vulnerability assessment of systems software," 2005, pp. 615–620.

[7] P. Li, M. Shaw, and J. Herbsleb, "Selecting a defect prediction model for maintenance resource planning and software insurance," *EDSER-5 Affil. ICSE*, pp. p32-37, 2003.

[8] H. Joh and Y. K. Malaiya, "Modeling Skewness in Vulnerability Discovery: Modeling Skewness in Vulnerability Discovery," *Qual. Reliab. Eng. Int.*, vol. 30, no. 8, pp. 1445–1459, Dec. 2014.

[9] G. R. Hudson, "Program errors as a birth-and-death process," System Development Corp., Report SP-3011, Dec. 1967.

[10] R. Anderson, "Security in open versus closed systems—the dance of Boltzmann, Coase and Moore," Cambridge University, England, Technical report, 2002.

[11] O. H. Alhazmi and Y. K. Malaiya, "Modeling the vulnerability discovery process," in *16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)*, 2005, pp. 10 pp. – 138.

[12] J. Kim, Y. K. Malaiya, and I. Ray, "Vulnerability Discovery in Multi-Version Software Systems," in *10th IEEE High Assurance Systems Engineering Symposium, 2007. HASE '07*, 2007, pp. 141–148.

[13] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam, "Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems," in *Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering*, New York, NY, USA, 2004, pp. 263–272.

[14] S. Woo, O. Alhazmi, and Y. Malaiya, "Assessing Vulnerabilities in Apache and IIS HTTP Servers," 2006, pp. 103–110.

[15] O. H. Alhazmi and Y. K. Malaiya, "Application of Vulnerability Discovery Models to Major Operating Systems," *IEEE Trans. Reliab.*, vol. 57, no. 1, pp. 14–22, Mar. 2008.

[16] F. Massacci and V. H. Nguyen, “An Empirical Methodology to Evaluate Vulnerability Discovery Models,” *IEEE Trans. Softw. Eng.*, vol. 40, no. 12, pp. 1147–1162, Dec. 2014.
[17] A. K. Shrivastava, R. Sharma, and P. K. Kapur, “Vulnerability discovery model for a software system using stochastic differential equation,” in *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Greater Noida, India, 2015, pp. 199–205.
[18] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, “Cluster-based vulnerability assessment of operating systems and web browsers,” *Computing*, Sep. 2018.
[19] Y. Movahedi, “Some Guidelines for Risk Assessment of Vulnerability Discovery Processes,” University of Maryland, 2019.
[20] Z. Rajabi, A. Shehu, and H. Purohit, “User Behavior Modelling for Fake Information Mitigation on Social Web,” in *Social, Cultural, and Behavioral Modeling*, vol. 11549, R. Thomson, H. Bisgin, C. Dancy, and A. Hyder, Eds. Cham: Springer International Publishing, 2019, pp. 234–244.
[21] Y. Roumani, J. K. Nwankpa, and Y. F. Roumani, “Time series modeling of vulnerabilities,” *Comput. Secur.*, vol. 51, pp. 32–40, Jun. 2015.
[22] L. Wang, Y. Zeng, and T. Chen, “Back propagation neural network with adaptive differential evolution algorithm for time series forecasting,” *Expert Syst. Appl.*, vol. 42, no. 2, pp. 855–863, Feb. 2015.
[23] N. R. Pokhrel, H. Rodrigo, and C. P. Tsokos, “Cybersecurity: Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach,” *J. Inf. Secur.*, vol. 08, no. 04, pp. 362–382, 2017.
[24] Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi, “Cluster-based Vulnerability Assessment Applied to Operating Systems,” presented at the 13th European Dependable Computing Conference, Geneva, Switzerland, 2017.
[25] A. A. Younis, H. Joh, and Y. Malaiya, “Modeling Learningless Vulnerability Discovery using a Folded Distribution,” in *Proceedings of the International Conference on Security and Management (SAM)*, 2011, pp. 617–623.
[26] O. H. Alhazmi and Y. K. Malaiya, “Measuring and Enhancing Prediction Capabilities of Vulnerability Discovery Models for Apache and IIS HTTP Servers,” in *2006 17th International Symposium on Software Reliability Engineering*, 2006, pp. 343–352.
[27] L. Allodi, “The Heavy Tails of Vulnerability Exploitation,” in *Engineering Secure Software and Systems*, 2015, pp. 133–148.
[28] T. Y. Yang and L. Kuo, “Bayesian computation for the superposition of nonhomogeneous poisson processes,” *Can. J. Stat.*, vol. 27, no. 3, pp. 547–556, Sep. 1999.
[29] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, “Comparison of ARIMA and Artificial Neural Networks Models for Stock Price Prediction,” *J. Appl. Math.*, vol. 2014, pp. 1–7, 2014.
[30] C. Bennett, R. A. Stewart, and C. D. Beal, “ANN-based residential water end-use demand forecasting model,” *Expert Syst. Appl.*, vol. 40, no. 4, pp. 1014–1023, Mar. 2013.
[31] N. Kourentzes, D. K. Barrow, and S. F. Crone, “Neural network ensemble operators for time series forecasting,” *Expert Syst. Appl.*, vol. 41, no. 9, pp. 4235–4244, Jul. 2014.
[32] A. Aslanargun, M. Mammadov, B. Yazici, and S. Yolacan, “Comparison of ARIMA, neural networks and hybrid models in time series: tourist arrival forecasting,” *J. Stat. Comput. Simul.*, vol. 77, no. 1, pp. 29–53, Jan. 2007.
[33] H. G. Hosseini, D. Luo, and K. J. Reynolds, “The comparison of different feed forward neural network architectures for ECG signal diagnosis,” *Med. Eng. Phys.*, vol. 28, no. 4, pp. 372–378, May 2006.
[34] D. N. Gujarati and D. C. Porter, *Basic Econometrics*. McGraw-Hill Irwin, 2009.
[35] R. May, G. Dandy, and H. Maier, “Review of input variable selection methods for artificial neural networks,” in *Artificial neural networks-methodological advances and biomedical applications*, InTech, 2011.
[36] S. Siami-Namini and A. S. Namin, “Forecasting Economics and Financial Time Series: ARIMA vs. LSTM,” *ArXiv Prepr. ArXiv180306386*, 2018.
[37] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, “Forecasting with artificial neural networks:,” *Int. J. Forecast.*, vol. 14, no. 1, pp. 35–62, Mar. 1998.
[38] L. Mentaschi, G. Besio, F. Cassola, and A. Mazzino, “Problems in RMSE-based wave model validations,” *Ocean Model.*, vol. 72, pp. 53–58, Dec. 2013.
[39] S. R. Hanna, D. W. Heinold, A. P. I. H. and E. A. Dept, and E. R. & T. Inc, *Development and application of a simple method for evaluating air quality models*. American Petroleum Institute, 1985.
[40] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, “Discovering physical concepts with neural networks,” *ArXiv Prepr. ArXiv180710300*, 2018.
[41] V. H. Nguyen, S. Dashevskyi, and F. Massacci, “An automatic method for assessing the versions affected by a vulnerability,” *Empir. Softw. Eng.*, vol. 21, no. 6, pp. 2268–2297, Dec. 2016.
[42] B. Littlewood *et al.*, “Towards Operational Measures of Computer Security,” *J. Comput. Secur.*, vol. 2, no. 2–3, pp. 211–229, Jan. 1993.